Polytechnic University of Catalonia

Checkpoint delivery

OpenSlicer

A portable three-dimensional object slicer

Author
Luuk Willemsen

Director
Marta Fairén

June 27, 2019

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Contents

1 Context and scope

1.1 Introduction

1.1.1 Project goal

There are multiple techniques to create real-world objects from a three-dimensional model. We will focus on Fused Filament Fabrication, a specific form of 3D printing, which consists in melting a filament of thermoplastic polymer, in a layer-by-layer fashion.

When modeling 3D objects with computer programs, they are usually represented as a triangle mesh (for example, in STL format). A 3D printer can only execute simple commands, such as moving a motor forward, or extruding filament. These commands are usually in GCODE format.

Slicing is the process of converting the 3D model as a triangle mesh to the GCODE required by the 3D printer. A 3D printer typically has a simple microcontroller that is not powerful enough to slice the model, so an external program, called a SLICER is used.

It is useful for 3D printer owners to remotely monitor and control it, stop it in case of a failure, and even start a print while not being present. This is usually accomplished by connecting a small computer to the printer. These computers are able to slice a model, but often have limited resources, difficult processor architectures, or the process is very slow.

To solve this, we create OpenSlicer. A slicer that is designed to run in a web browser, so it is platform independent. Since the slicing occurs in the client's browser, the slicer can easily be hosted as a static page on the 3D printer itself.

1.1.2 Stakeholders

- **Project developer**. I will be developing the project, doing the necessary research, testing and programming to make it succeed.
- **Project director**. Marta Fairén is the director of this project. As an Associate Professor of the Computer Science department of the Polytechnic University of Catalonia, she will be helping the project devel-

oper with any problems, as well as providing feedback, improvement suggestions and technical advice to make sure the project succeeds.

- **The community** will be able to contribute, learn, improve, share, and use the software.
- **3D printer owners** will be able to use the slicer from their phones or laptops to remotely slice their objects and print them.
- **3D printer manufacturers** will be able to host the slicer on the printer directly, without having to worry about updates, or maintenance of the slicer. They can then focus only on their main business: building 3D printers.

1.2 State of the art

In this section we will briefly discuss existing technology, as well as what this project aims to accomplish.

1.2.1 3D viewers

3D visualization of models is a deeply explored field, and is used in all kinds of software: games, design studios, CAD programs, medical applications, etc. There are many robust libraries that offer 3D visualization, like DirectX, Metal or OpenGL. These libraries however are very low level, and too tedious to work with for our purpose. Instead, we will use **three.js** [1], a high-level 3D visualization library for the web.

1.2.2 Polygon clipping

There are numerous clipping libraries in different languages [2, 3, 4]. This field has been deeply explored in the past, and efficient polygon clipping is considered a solved problem. We will use **clipper-lib** for polygon clipping.

1.2.3 Slicers

Some slicers already exist in the literature [5, 6]. However, there are no slicers in JavaScript. Therefore, this project will use mostly own algorithms but some will be inspired on existing slicers or ports of well-known slicing algorithms for other language.

1.2.4 Bridging the gap

Existing slicers are designed for optimal performance, and are compiled to run as a binary on the user's system. We compromise performance, as our slicer will run in the browser, but obtain the following advantages:

- A 3D printer's microcontroller can serve the slicer without the need of a more powerful computer.
- Slicing on mobile phones and tablets
- Refreshing the page will update the slicer.
- Browser based, so cross platform by nature
- No dependencies or installs.
- Possibility to run in the cloud and manage 3D models from any device.

Most of these features are new for Fused Filament Fabrication slicers. There is a performance overhead of running the software in a browser, but it is compensated for by a possible cloud-based slicing engine (if we decide to monetize this idea), and by the availability on all platforms.

1.3 Scope

A fully functional slicer will be developed and tested on a live 3D printer. The slicer should be able to output GCODE that produces a physical object that accurately resembles the original 3D model.

1.3.1 Minimum requirements

There are several stages to the slicing process that must be implemented for it to work properly:

- **A graphical user interface** to move, scale and rotate the object, as well as manipulating different settings of the slicing process, such as the speed, layer height, nozzle diameter, print and bed temperature.
- **A 2-manifold [7] check algorithm**, that detects whether the input model is watertight and can be sliced correctly.
- **A 3D geometry intersection algorithm**, that generates layer perimeter polygons intersecting a plane with the model's triangles.
- **A 2D polygon clipping algorithm** to generate internal perimeters and calculate the infill.
- **An ordering algorithm** to create a chain of segments that will form each layer in the final print. This should minimize the total print time.
- **An output generator**, that will take the segment chain and convert it to GCODE for printing. This generator might also add calibration, homing, or other instructions at the begin or end of the gcode.

1.3.2 Extras

Additional features that greatly improve print quality, but are harder to implement may be added if time permits:

- **Infill pattern generation**, to avoid printing at 100% density.
- **Automatic support structure generation**, to be able to print with overhangs correctly.
- **Decimation**, to allow extremely detailed objects to be sliced correctly.
- **Retraction**, to improve bridging and reduce stringing.
- **Layer-by-layer settings** can greatly improve surface finish while keeping print times low.

1.4 Possible obstacles and solutions

This section will discuss some of the problems that might arise during the development of the project.

- **Breaking the 3D printer** would leave us without a testing machine. This is a realistic problem because sending the wrong GCODE to the printer may run the nozzle into the bed, or heat it so much that the power supply fails.
- **Floating point precision**. The precision of floating points or doubles is limited, so we will need to find a solution to that. There are several ways to do this, which will be discussed in the technical part of this project.
- **Bugs.** Developing a slicer is not an easy task. Companies have been developing them for decades and their commercial products are still containing a lot of bugs. It seems natural that our team consisting of a single developer will have some trouble designing a fully functional slicer from scratch and will encounter some time-consuming bugs that might lead to delays or even missed deadlines. We will use a version control system and continuous integration tests to track bugs and prevent them from happening in the first place.
- **Algorithm efficiency**. It is entirely possible that the algorithms developed during this project are not fast enough to run on decent hardware. If this is the case the project might fail completely, or be too slow to be usable. The solution to this problem is to ensure that we are using reasonably fast algorithms.

1.5 Methodology and rigor

1.5.1 Methodology

Since there will be a single developer in the project, it does not make sense to follow very complex methodologies, but we will apply some good practices and principles from well-known methodologies:

- From **Test Driven Development**: We will have automated tests before every commit to avoid introducing most bugs.

- From **Agile Development** We will only commit code that compiles and runs correctly, and make small changes.
- We will set some milestones in order to be aware of the general timeline of the project.
- Using git as version control system will allow us to go back to any previous state of the project, as well as to rapidly track down bugs using git-bisect [8]

1.5.2 Rigor

- To check whether our project meets industry standards we will benchmark the different stages of the slicing process.
- We will exhaustively test edge cases and difficult objects in order to validate the robustness of our slicer. This will be done in an automated fashion as well as by hand.
- Real 3D prints will be used to measure the accuracy of the slicer, absolute and relative errors will be measured and improved on.

2 Project planning

2.1 Schedule

2.1.1 Estimated duration

The work required for this project, including development, documentation, possible bugs and fixes, is approximately 4 months.

All possible issues mentioned in deliverable 1 are taken into account when designing this planning. However, during project development it might be possible that the proposed planning is altered.

2.2 Task description

This project has a lot of different modules that are independent from each other in functionality, but some might require the output of a previous module before being able to work properly. The planning is important to follow, so we do not break time dependencies between the modules.

2.2.1 Initial 3D Viewer and GUI

We will use *three.js* as a base for the 3D viewer and will work on top of it. It's fairly easy to set up and show any 3D model in the viewer, as it's one of the basic examples included in the library. We then need some functions to rotate, scale, translate and otherwise manipulate the object. I have some experience with three.js, so this task should last no longer than a week. This task requires the use of the mentioned library, as well as an IDE. We will use WebStorm [9] for the development of the project and all of it's components.

2.2.2 Slicer

Sharing the name with the overall process, in this case the slicing algorithm will only be responsible for obtaining a polygon for a given height. This module will be called multiple times in order to slice the whole object.

2.2.3 Toolpath generator

After the slicer has generated a polygon, this module will be responsible for generating a set of lines in an internal format that will be used by the *GCODE* generator later on.

2.2.4 GCODE generator

Once we have our toolpath generated, we need to convert it to a language the 3D Printer understands: GCODE. This module is responsible for the conversion and the final output of the program. This task requires a 3D printer or simulator to test the generated output.

2.2.5 Resources

The following resources are required in order to complete the previous tasks.

- All modules require *human resources* to develop the required code.
- A computer to develop with.
- A 3D printer or simulator to test the output of the slicer.
- A place for the developer to work
- All the mentioned software

2.3 Time estimation

An estimation of the project's timeline is shown in Table 1. Times are approximate but should account for most possible changes and delays.

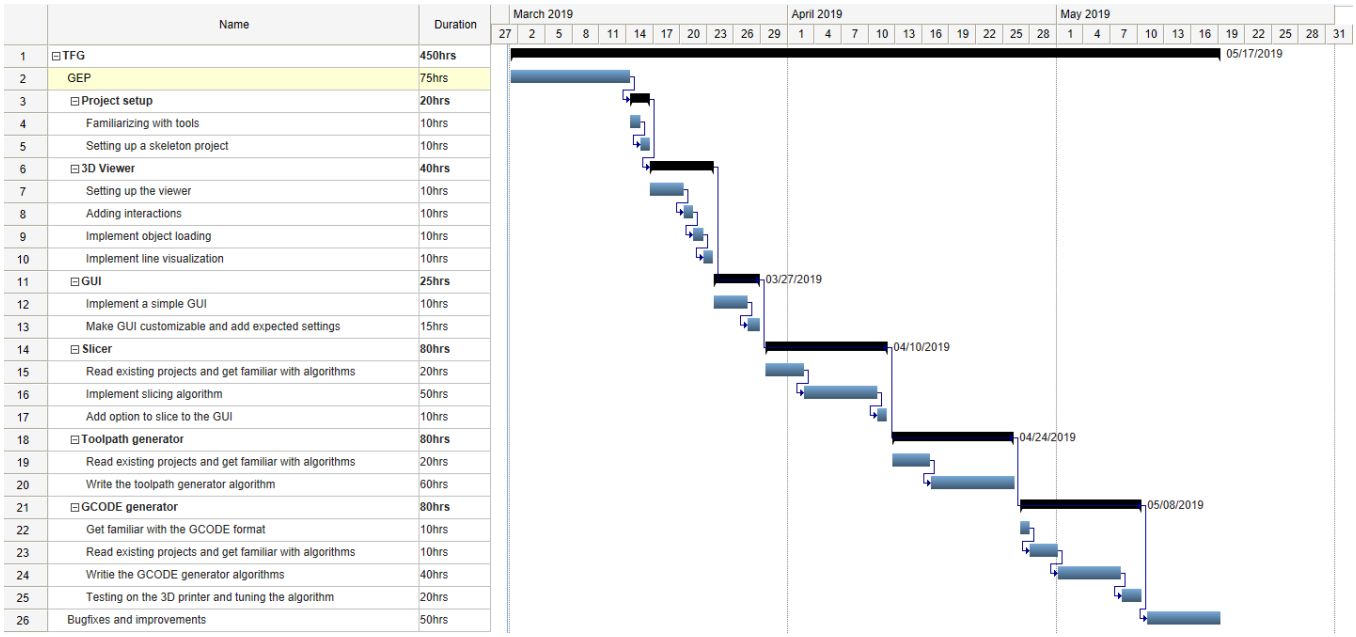
Task	Estimated duration (h)
GEP	75
Project setup	20
3D Viewer	40
GUI	25
Slicer	80
Toolpath generator	80
GCODE generator	80
Bugfixes and improvements	50
Total	450

Table 1: Estimated time for each task

Every task or module will have several stages. We will need to read through existing examples and projects to get a general idea of what to do. Then we will need to decide the implementation details, implement the module, test it and fix any possible bugs.

2.3.1 Gantt chart

Figure 1 shows a more detailed overview of the task planning in a Gantt chart. The timeline may change slightly due to project requirements or unexpected bugs. This is accounted for in the final stage of the project, bugfixes and improvements. More on this in the next section.



	Name	Duration
1	TFG	450hrs
2	GEP	75hrs
3	Project setup	20hrs
4	Familiarizing with tools	10hrs
5	Setting up a skeleton project	10hrs
6	3D Viewer	40hrs
7	Setting up the viewer	10hrs
8	Adding interactions	10hrs
9	Implement object loading	10hrs
10	Implement line visualization	10hrs
11	GUI	25hrs
12	Implement a simple GUI	10hrs
13	Make GUI customizable and add expected settings	15hrs
14	Slicer	80hrs
15	Read existing projects and get familiar with algorithms	20hrs
16	Implement slicing algorithm	50hrs
17	Add option to slice to the GUI	10hrs
18	Toolpath generator	80hrs
19	Read existing projects and get familiar with algorithms	20hrs
20	Write the toolpath generator algorithm	60hrs
21	GCODE generator	80hrs
22	Get familiar with the GCODE format	10hrs
23	Read existing projects and get familiar with algorithms	10hrs
24	Writie the GCODE generator algorithms	40hrs
25	Testing on the 3D printer and tuning the algorithm	20hrs
26	Bugfixes and improvements	50hrs

Figure 1: Gantt chart of the project. Generated with Gantter [10]

2.4 Alternatives and action plan

In the previous delivery we mentioned the methodologies that will be used, including Test Driven Development, and heavy use of git. This should reduce the risk of bugs and accelerate the development quality as well as the speed.

We have a total workload of 450 hours, with a dedication 40 hours per week. Extra time is given at the end to account for possible bugs or potential problems and slowdowns, which are discussed in the previous delivery.

The 50 hours of bug fixes and improvements are not necessarily at the end. Instead they are distributed throughout the project. In case of a problem, the complete focus will be on solving it as soon as possible, and this time will be deducted from the bug fix time at the end. Any time left at the end

will be used for improvements.

Particular detailed are discussed as follows:

2.4.1 Breaking the 3D printer

We will mitigate the risk of breaking the 3D printer by running wrong gcode on it by simulating it first. Slic3r and Cura both have a gcode viewer, so we can verify that the generated output is correct (or at least it won't move to negative coordinates, or outside of the build area). In case the printer ends up breaking, we have an alternative, cheaper 3D printer available for testing.

2.4.2 Floating point precision

Floating point precision may not be enough for some geometry operations. In these rare cases, the model will still be sliced correctly, but will have some features or some missing detail. This is a limitation of all slicers, and it is possible to still get those detail, but no current slicer does this so we consider this beyond the scope of this project.

2.4.3 Bugs

We already discussed multiple times how we will deal with possible bugs. In particular, whenever a bug is found, we will write a test case that reproduces the bug, and use git bisect to find the exact commit where the bug was introduced. Since we are also making small commits with specific changes, it will be easy to find out what the cause of the bug was and how to fix it.

2.4.4 Algorithm efficiency

Some of the slowest algorithms are already handled by libraries, but we still need to find efficient algorithms for slicing, polygon generation and toolpath generation. We have assigned extra time to each of these module's planned time to account for finding efficiency improvements. Since most slicers are able to run on slow hardware, it is very likely that every algorithm will be possible to implement with acceptable efficiency. Alternatively we can accept

slower algorithms that will still produce a decent running time, although it will be noticeable to the user.

3 Budget and sustainability

3.1 Project budget

In this section, an estimation of the cost of the project is presented, taking into account the previously mentioned resources.

3.1.1 Hardware budget

Table 2 shows the hardware resources budget needed in order to develop and test the project.

Product	Price	Units	Useful life	Amortization
Apple MacBook Pro	2000€	1	8 years	250€
Prusa i3 MK3 3D Printer	769€	1	4 years	192.25€
Total	2769€			442.25€

Table 2: Hardware budget

3.1.2 Software budget

Table 3 shows the software resources budget needed in order to develop and test the project.

Product	Price	Useful life	Amortization
Git	0€		0€
GitHub	0€		0€
macOS High Sierra	0€		0€
Google Chrome browser	0€		0€
Gantter	0€		0€
Slic3r	0€		0€
Slic3r PE	0€		0€
Ultimaker Cura	0€		0€
LaTeX	0€		0€
webpack	0€		0€
WebStorm	129€	1 year	129€
Total	129€		129€

Table 3: Software budget

3.1.3 Human resources budget

Table 4 shows the human resources budget needed in order to develop and test the project.

Role	Price/h	Hours	Cost
Project manager	50€	50	2,500€
Software developer	30€	300	9,000€
Tester	30€	100	3,000€
Total		450	14,500€

Table 4: Human resources budget [11, 12, 13]

Since the modules of the project are mostly independent, and equally important, the different roles are distributed evenly across all tasks.

3.1.4 Unexpected costs

Table 5 shows a generous budget margin for problems or deviations that may arise. This extra budget is very unlikely to be used because of the robust planning of the previous deliverable.

Role	Price/h	Hours	Cost
Project manager	50€	10	500€
Software developer	30€	20	600€
Tester	30€	10	300€
Total			1,400€

Table 5: Extra human resources budget [11, 12, 13]

3.1.5 Direct costs breakdown

Table 6 shows the breakdown of costs per task listed in the Gantt chart.

Task	HW	PM	SW Dev	Tester	Total
Project Setup	571.25€	250€	1500€	0€	1879€
3D Viewer, GUI	0€	500€	1500€	600€	2,600€
Slicer	0€	500€	1500€	600€	2,600€
Toolpath generator	0€	500€	1500€	600€	2,600€
GCODE generator	0€	500€	1500€	600€	2,600€
Bugfixes	0€	250€	1500€	600€	2,350€
Total	571.25€	2,500€	9,000€	3,000€	15,071.25€

Table 6: Direct costs breakdown

3.1.6 Indirect costs

Table 7 shows the indirect costs related to developing and testing the project.

For this project we will fix the price of electricity at 0.12€/kWh [14]. A 3D printer consumes about 0.2kWh, a laptop about 0.06kWh, and the rest (router, lights) about 0.03kWh. We will assume everything is running during the whole development of the project, except for the 3D printer, which will only be needed in the latest stages of the project, and we will consider it is used only the last 50% of the project. This brings us at a total electricity consumption of about 100kWh.

Product	Price	Amount	Cost
Electricity	0.12€/kWh	100kWh	12€
Internet	30€/month	4 months	120€
Coffee and snacks	10€/week	16 weeks	160€
Total			292€

Table 7: Indirect costs [14, 15, 16]

3.1.7 Total budget

We will calculate the total budget by adding all previous budgets together. This is shown in Table 8.

Concept	Estimated cost
Hardware	442.25€
Software	129€
Human resources	14,500€
Unexpected costs	1,400€
Indirect costs	292€
Total	16,763.25€

Table 8: Total budget

3.1.8 Budget management

It is unlikely that we will exceed our budget, since our project planning section was quite robust. We have already accounted for a generous error margin to fix bugs or compensate for delays in the development of complex algorithms. It might still be possible to need extra resources or hire an additional software developer as a consultant. We have also taken this into account when creating our budget. We are being extremely cautious about human resources budget allocation since this is the most expensive part of it.

As we discussed earlier, it is possible to break the 3D printer while doing our testing. In this case we will work on a gcode simulator that comes with most existing slicers. This will add nothing to our budget, and we will continue the project in a theoretical way, without actual 3D printed tests or examples.

We will be monitoring the progress of the project according to the Gantt chart shown previously. If delays occur, we will allocate an extra software developer or tester to solve the specific issue that is causing the delay. In this case, this extra developer will count against the extra budget we have defined earlier.

Following the previous procedure, we can guarantee that we will not exceed our timeline significantly, and we will still stay in our allocated budget, since we already accounted for possible delays or unexpected costs.

3.2 Sustainability report

In this section we analyze the sustainability of the project. Its impact is analyzed in its three dimensions: social, economical and environmental.

This is a project consisting of mostly software. There are alternatives already in the market with similar features. We will not be causing a significant social, economical or environmental impact.

Sustainability does not really apply to this project in all of its dimensions, but we will analyze them anyway for the sake of completeness.

3.2.1 Social impact

On a personal level, this project allows me to push my limits on development and testing, as well as to deduce complex algorithms and development methods. It will help me know my limits and capabilities and provide me with a fun time to learn and experiment.

To the end user, this project does not have a significant social impact, since it is only a tool to convert solid models to 3D printer instructions.

3.2.2 Economical impact

We have already discussed and quantified the budget of this project in terms of hardware, software, and human resources.

We try to use mostly free software, and our product will also be a free software tool. Existing free software slicers exist already, so this project will not make

a big economical impact on the end user.

3.2.3 Environmental impact

We will be using 100kWh of electricity while developing and testing the project, which is equivalent to approximately 28.3 kg of CO_2. This might seem like a lot until we add our team of humans, who breathe during development, which accounts for approximately 50 more kg of CO_2, for a total of 78.3 kg [17].

We will be using Polylactic Acid (PLA) as our only plastic during the testing phase. It is a bio-plastic material, biodegradable and reduces CO_2 [18, 19]. This is one of the best materials to use when 3D printing and we choose this for our testing because of its nice environmental properties.

3.2.4 Sustainability Matrix

Taking into account the previous sections, in Table 9 we show the sustainability matrix of this project in its three dimensions.

	PPP	Useful Life	Risks
Environmental	Design consumption	Ecological footprint	Environmental risks
	9 / 10	18 / 20	-5 / -20
Economical	Bill	Viability plan	Economical risks
	8 / 10	17 / 20	-2 / -20
Social	Personal impact	Social impact	Social risks
	8 / 10	10 / 20	-1 / -20
Sustainability range	25 / 30	45 / 60	-8 / -60
	62 / 90		

Table 9: Sustainability matrix

4 Legal

4.1 License

4.2 Laws and regulations

This project is published as free software (see license section above) under a MIT license (or similar).

Since this is a software-only project, the above license covers all legal use of the software, and explicitly waives all liability and possible claims for the author. Users are responsible for any use of the software.

5 Technical skills

The technical skills that are considered relevant in this projects are the following, as per the project's initial presentation:

- **CCO1.1**: Evaluate the computational complexity of a problem, learn algorithmic strategies that can lead to its resolution, and recommend, develop and implement the one that guarantees the best performance in accordance with the established requirements.
- **CCO2.3** Develop and evaluate interactive and complex information presentation systems, and their application to solving computer interaction design problems.
- **CCO2.6** Design and implement graphics, virtual reality, augmented reality and video games applications.

6 User Interface

The User Interface consists of various components. In this section we will explain their functionality. A full sample screenshot of the slicer is shown in Figure 2.

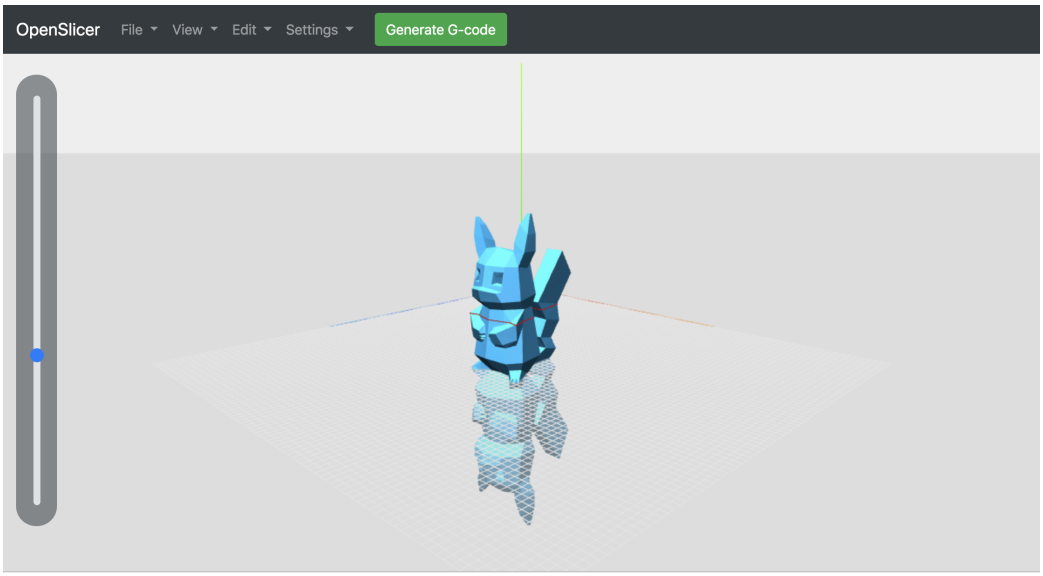


Figure 2: OpenSlicer: Sample screenshot

6.1 3D Viewer

The 3D Viewer is responsible for displaying the model and the user's interaction with it. It will also display the paths preview once the model is sliced.

We use *three.js* as a *WebGL* [20] abstraction layer. This library is very well tested and makes it extremely easy to manipulate 3D objects. The included *OrbitControls* [21] allows for easy user interaction, like camera zooming, rotating and panning.

We will use the *STL Loader* [22], to load any STL model as a three.js object, as shown in this demo.

Figure 3 shows an example of the 3D Viewer with some basic models.

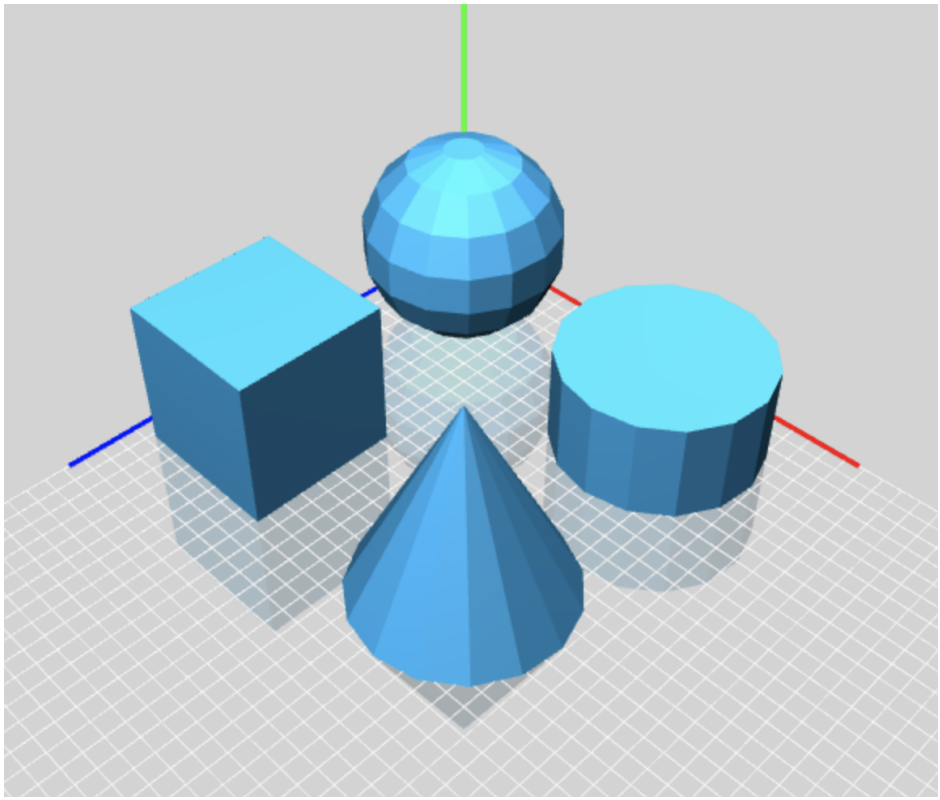


Figure 3: 3D Viewer example: Simple objects

The 3D Viewer has the following elements in the scene:

- **Model**: The model itself, once it is loaded
- **Build plate**: A simple plane that represents the printer's build plate. The model will be centered on it, unless the user moves it later.
- **Axes helper**: Since WebGL uses the Y-up axes configuration, while 3D printers normally use a Z-up configuration, we show the axes to avoid any confusion.
- **Point lights**: We have three point lights, located at `[50, 200, -100]`, `[-0, 200, 200]` and `[200, 300, 250]`.
- **Mirrored model**: The model below the build plate is a mirrored version of the original model. The build plate has some transparency, creating a good looking finish without distracting the user too much.

6.2 GUI

We will be using a 2D graphical user interface to allow the user to fine-tune the settings. This is important because of the great variety in 3D printer models that come in different sizes, have different speeds, materials, etc.

This GUI will control both printer-related settings (layer height, speed, temperature, etc), as well as the 3D Viewer settings (object color, rotation, translation, scale, etc).

We choose *Bootstrap 4* [23] for its ease of use and flexibility. We will use a Navbar with dropdown menus and sub-menus to configure all options related to the viewer, as well as to the printer.

Figure 4 shows an example menu using Bootstrap with some of the settings that we will implement.

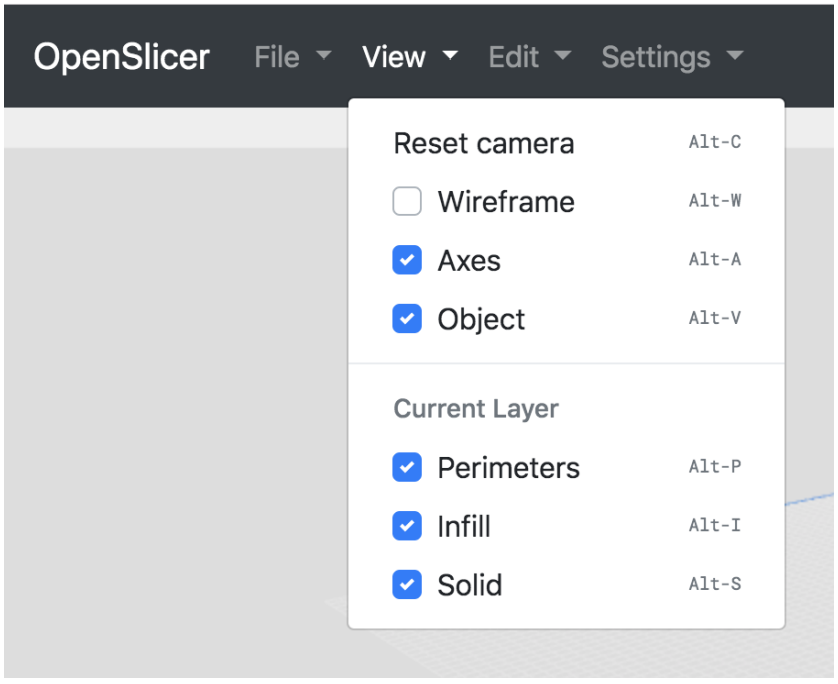


Figure 4: Bootstrap 4: View Menu

6.2.1 The File menu

The file menu contains all operations related to opening, processing and exporting files.

- **Open**: Opens a dialog where the user can select a file.
- **Slice**: Once a model is loaded, this option runs the slicing algorithm and displays the result in the 3D Viewer. The slicing part does not generate the G-code and only shows a preview in WebGL.
- **Generate G-code** This will generate the G-code once the model is sliced. It will also download the file to the user's computer.

6.2.2 The View menu

The view menu contains all settings, configuration and actions related to the 3D Viewer.

Global View Settings relate to the scene and visibility of the whole model:

- **Reset Camera**: Resets the camera to look at the center of the build plate.
- **Wireframe**: Toggles the model between solid and wireframe modes. If wireframe is enabled, the build plate reflection is disabled.
- **Axes**: Shows or hides the XYZ axis.
- **Object**: Shows or hides the model. This option is useful while slicing, if we want to take a look at a single layer, we want to see complete slice, not only the perimeters. Hiding the model allows this.
- **Build plate**: Shows or hides the build plate.

Layer View Settings relate to the current layer.

- **Perimeters**: Shows or hides perimeters or outer shells of the object.
- **Infill**: Shows or hides infill that is not solid.
- **Solid**: Shows or hides solid infill, such as the bottom layer.

6.2.3 The Edit menu

The edit menu contains all operations to object manipulation.

- **Rotate**: Allows the user to edit the model's rotation. Rotation will be in degrees, as it is easier for a user to input. The angle will be converted to radians internally.
- **Translate**: Allows the user to edit the model's translation. Translation can only be performed on the X and Z axis, as the object will always be touching the build plate.
- **Scale**: Allows the user to edit the model's scale.

In each of these menus, a sub-menu will be opened to configure each individual axis as shown in Figure 5.

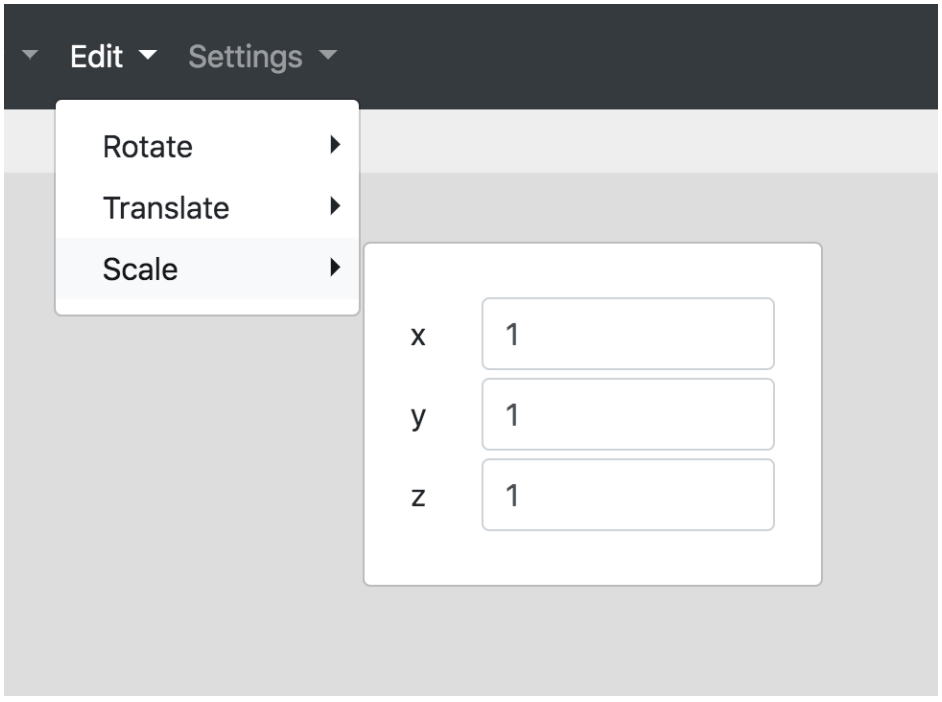


Figure 5: Edit -¿ Scale sub-menu

6.2.4 The Settings menu

The settings menu is the most important. It contains all settings related to the print job and printer. The settings menu is shown in Figure 6

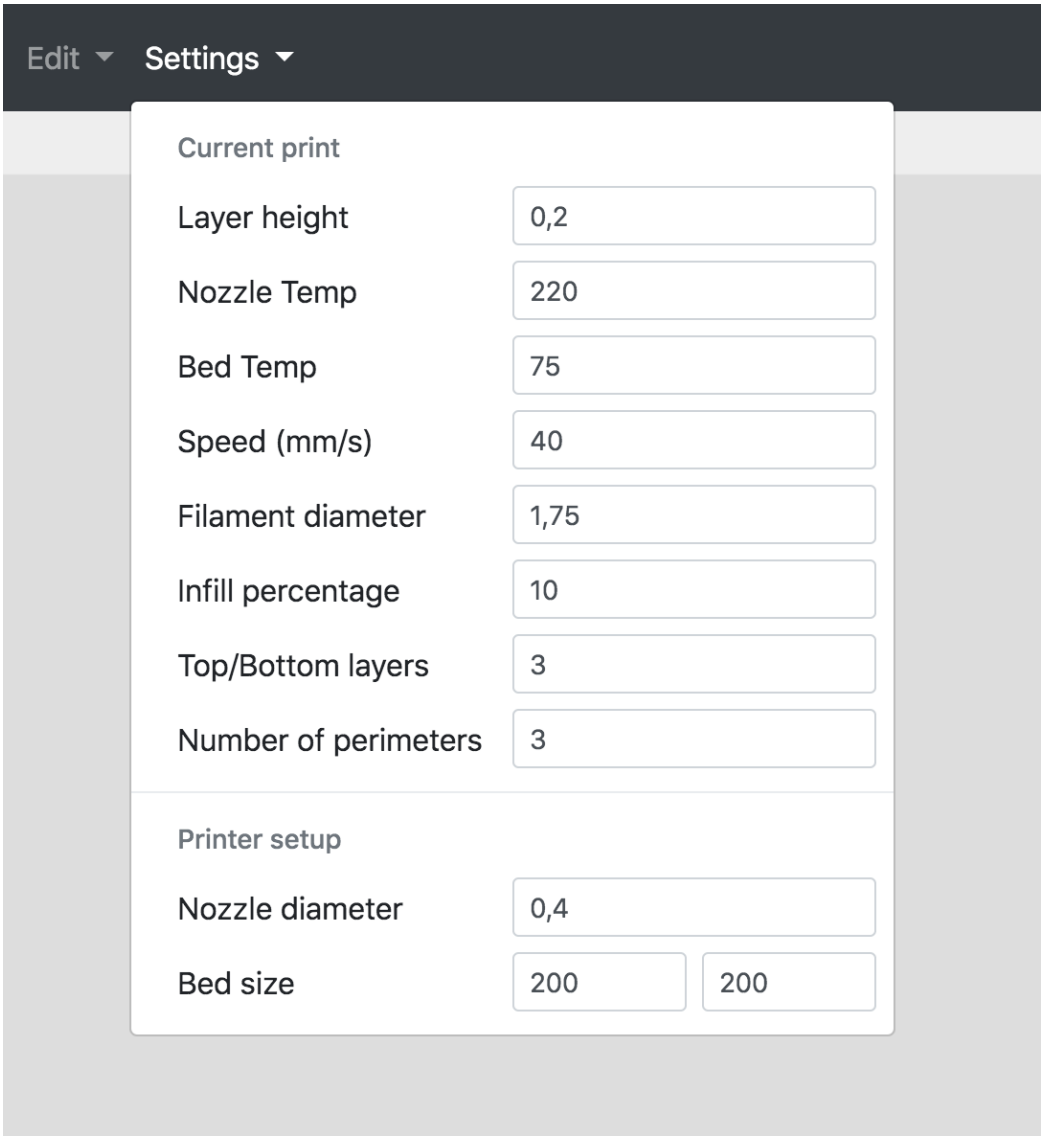


Figure 6: Settings menu

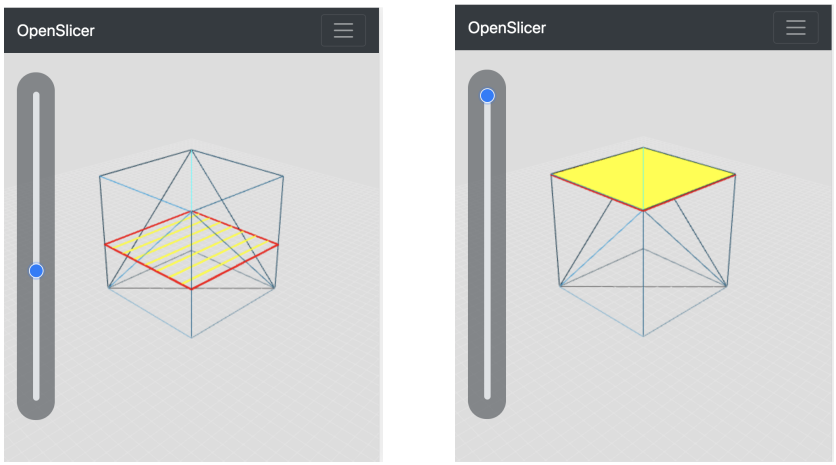
- **Layer height**: The resolution in the Z axis of the print. Increasing this setting will greatly reduce print time at the cost of quality.
- **Nozzle temperature**: This is the temperature at which the filament

will be exposed to once it is being pushed through the hot end and molten. For PLA, the typical values are between 180°C and 220°C.

- **Bed temperature**: Most high-end printers incorporate a heated bed to improve adhesion of the part. This setting allows to tune it's temperature.
- **Speed**: The speed at which the motors will travel while extruding material. This setting is normally set to 40mm/s but can go all the way up to 200mm/s for really fast prints. This setting will affect quality as well as wear on the mechanical parts of the printer.
- **Filament diameter**: Filament usually comes in spools of either 1.75mm or 3.00mm in diameter. We need to know it's size to calculate the length of the filament to extrude the correct amount of plastic.
- **Infill percentage**: To avoid wasting plastic, we do not need to print the inside of a model completely solid, and can instead print several different patterns to provide structure and strength while minimizing the amount of material needed to create the print.
- **Top/Bottom layers**: The amount of layers that we want solid, before we start generating infill patterns.
- **Number of perimeters**: The number of perimeters we want before starting to generate infill patterns.
- **Nozzle diameter**: Nozzles come in a variety of sizes. This setting will determine how wide the lines of the print are.
- **Bed size**: To calculate the center of the object and give the user a more realistic view of the size of the model in relation to the build plate.

6.2.5 Current Layer Slider

As shown in Figure 7, the slider on the right of the window selects the current layer to display. This allows us to easily scroll through the object and preview what the result will look like in a layer-by-layer fashion.



(a) Halfway infill slice (b) Top layer slice

Figure 7: Current layer slider

6.2.6 Keyboard shortcuts

Most of the settings shown in previous menus, especially view settings, have a keyboard shortcut. This allows the user to quickly perform actions or toggle what they want to see without navigating through the menus.

6.2.7 Data binding

We support one way data binding through listeners. That is, whenever a user changes a setting, it will be automatically updated in the internal state that we will use to slice the models.

7 Slicer

The Slicer is responsible for taking an object as a list of triangles, and return a list of layers that the Toolpath Generator can later translate into GCODE. Each of these layers contains a list of ordered segments, which represent the path for the printing head to travel along.

The slicing process is much easier to understand and implement if done in parts.

7.1 2-manifold checking

To verify that the object is actually printable, we need to check that it is a valid 2-manifold.

The initial plan was to test the validity of an object and rejecting non manifolds, but we have decided that it's best to just check and throw a warning in this case, and perform a slice on a best-effort basis. It is better to have an object slice wrong, than to not slice it at all.

Since this will only be a warning, it will be left out for now. In the final delivery it may or may not be included.

7.2 Slicer

It might be confusing to use the word Slicer for this part. Previously we defined slicing as the process of converting an object to 3D printer instructions. Slicing is also the process of cutting an object by a plane. This subsection is all about that: Intersecting a plane with a 3D object.

The algorithm we use for this is fairly simple: We simply take all triangles individually and intersect them with the plane. The intersection segment is added to the resulting list. If the plane intersects a given triangle on an edge or vertex, we will still include the segments in the result list.

This algorithm will return an unordered list of segments, that are the segments where the plane intersects the object. If the object is a valid 2-manifold, then the resulting list can be ordered in a unique way, constructing a valid Surface.

7.3 Polygon generator

The polygon generation consists of taking the unordered list of segments that the Slicer produced, and order them into a surface.

We define a surfaces to be a list of polygons and a list of holes, which are also polygons. This definition of surface can describe any boolean 2D geometry.

Provided the object was a valid 2-manifold, all vertices in the slicer's output have degree exactly 2, which makes it trivial to construct the list of polygons.

We still need to determine whether a polygon is a hole or not. We can determine this easily with a simple ray tracing: Given a polygon, we take one of its vertices v and trace a straight line in any direction. If it intersects an odd number of polygons on one side of v, then this polygon is a hole. Otherwise it's part of the surface.

For this algorithm we need to be careful with edge cases: If we happen to trace a line from a hole, that intersects a parent polygon on one of its vertices, we might count that intersection twice, and think that this polygon is part of the surface while it's actually a hole.

An example of the polygon generated by a horizontal slice of the model is shown in Figure 8 and Figure 9.

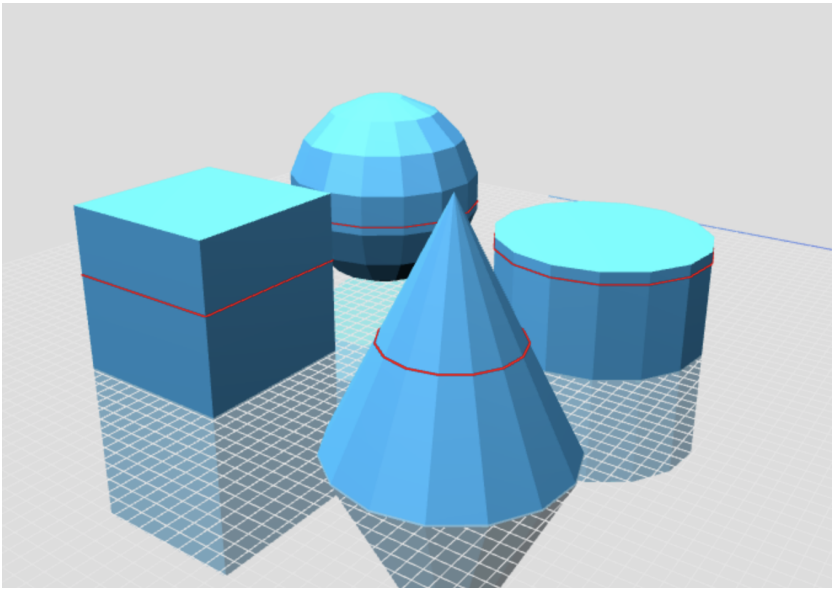


Figure 8: Slice: Intersection and object

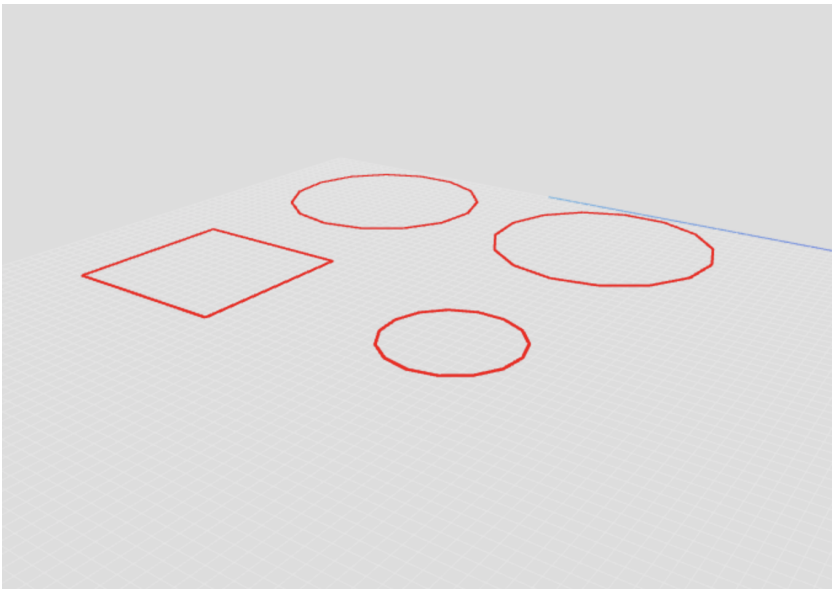


Figure 9: Slice: Intersection only

7.3.1 Why polygon generation?

Polygon generation is explained in this section but is not included in the final slicer due to the complexity of use of the clipping library we chose.

Instead of generating polygons for each layer, we can instead intersect the object directly with parallel lines for each desired height and width of the 3D printer. This will however generate only movement in one direction, and produces extremely ugly results, similar to when you color a drawing in a zigzag pattern instead of going around the edges first.

Having the polygon information allows us to create smaller versions of the polygons by perpendicularly offsetting them inwards by a fixed distance. This allows us to print the perimeters as a series of closed loops, and process the infill separately, without affecting the dimensions of the object, while also allowing for infill to be rectilinear.

7.4 Generating segments

We have two basic types of segments: Shell or perimeter segments, and solid or infill segments. The shell segments are the intersection of the plane with

the model, and the infill lines are generated as described below.

The segments will always be parallel lines, a fixed distance from each other. They will be clipped by the polygons generated in the previous step.

We will follow a similar approach than with the slicing process: We will intersect each polygon with vertical planes to obtain the segments we want to produce. This process is illustrated in Figure 10, where the intersection plane is displayed as a red line.

We intersect the polygon with each vertical plane, and obtain a list of co-linear points. We sort these points and create segments between every pair of points. This process is also illustrated in the same Figure 10, the points in blue, and the segments in green.

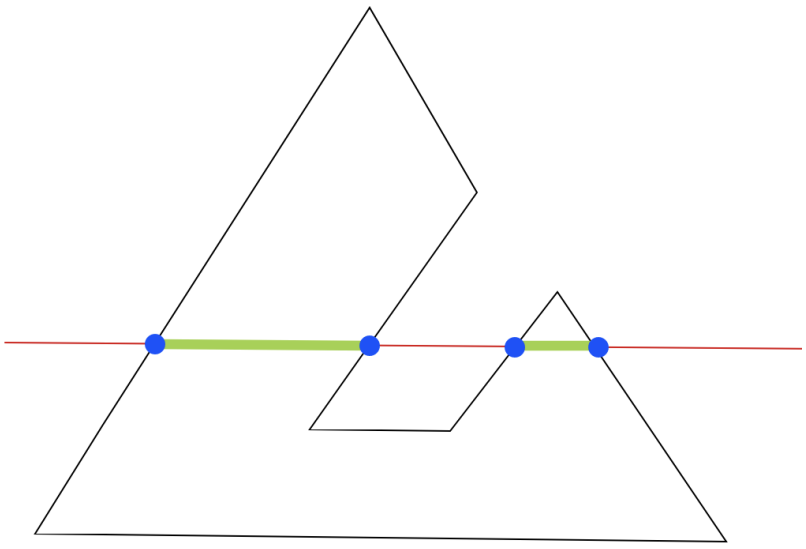


Figure 10: Solid infill segment generation

After this process, we end up with a list of segments that, if printed in the correct order, produce the expected object's infill.

An example with a Pikachu [24] is shown in Figure 11 and the resulting slice, shown in Figure 12, with the resulting segment lines in yellow and shell or perimeter segments in red.

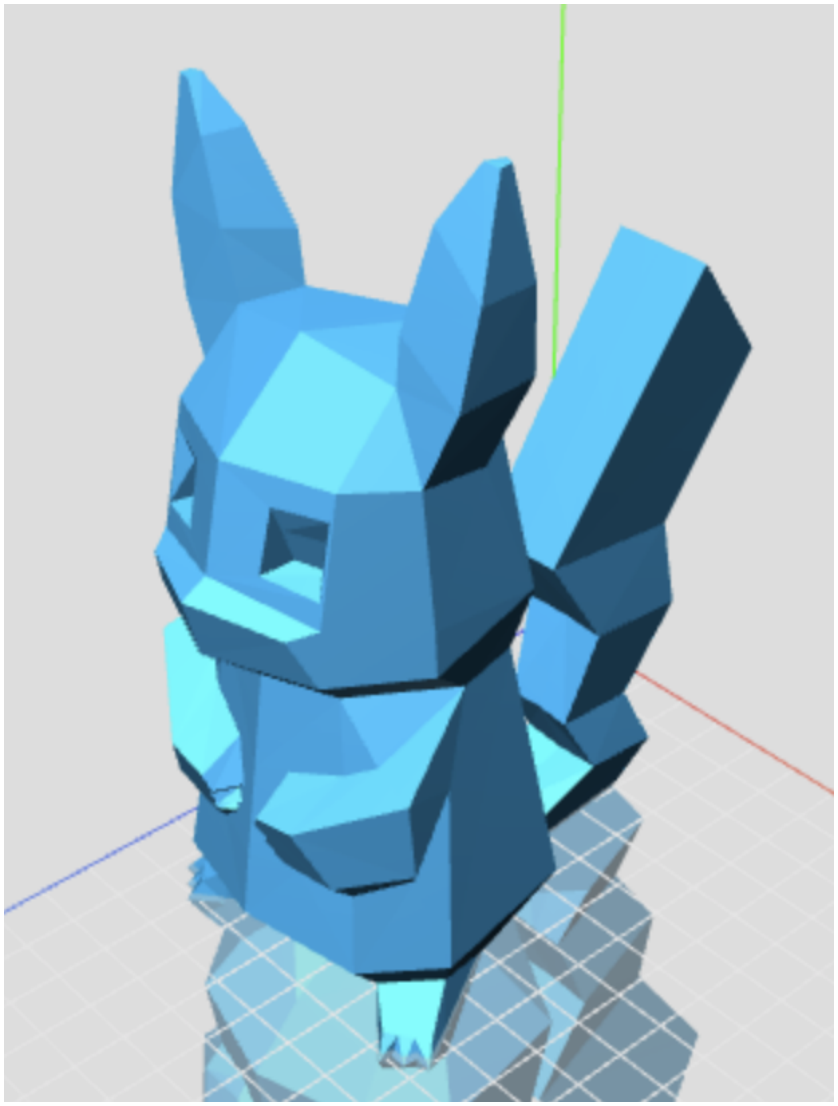


Figure 11: Pikachu ready to be sliced

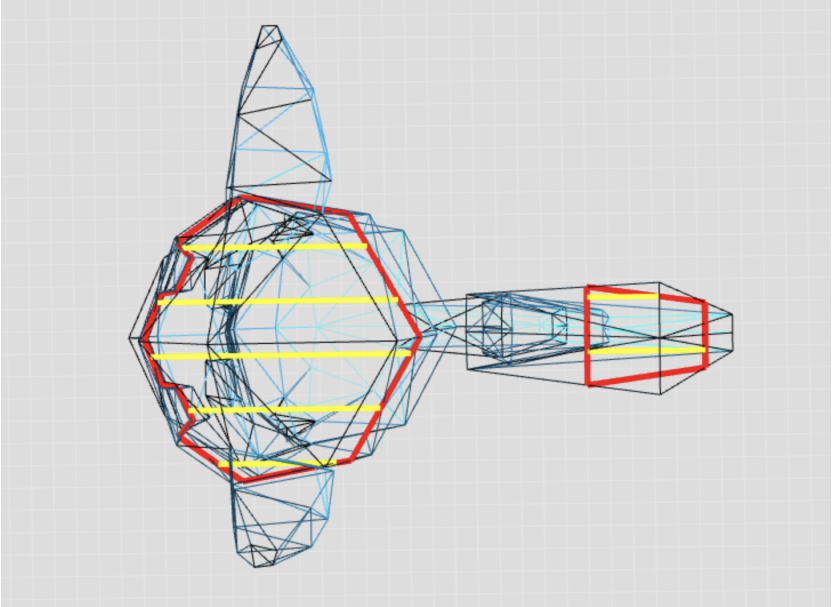


Figure 12: Top view of a single layer slice

You may have noticed the separation between the infill lines. This is called non-solid infill, an important concept to save time and material, which will be discussed next.

7.4.1 Generating non-solid infill

Generating a solid infill is both a waste of time and material. In most cases, the inner strength of a print will not require 100% infill, and we can greatly improve print speed and reduce material by changing the infill percentage. This is achieved by separating the infill lines by more than the line width. For example, if we skip every other line we will achieve an infill density of 50%. We can generalize this as shown in Table 10

Lines skipped	Infill density (%)
0	100
1	50
2	33
3	25
4	20
n	$100/(n+1)$

Table 10: Line skip - infill density

As a user we would like to specify the infill percentage though, so we will use the inverse function, $f(x) = 100/x - 1$, where x is the desired infill percentage to calculate how many lines to skip.

The final line pitch (distance between two adjacent lines) will be $w + w * f(x)$, where w is the line width, and x is the infill percentage.

If we specify a 0 infill density, we would get a division by zero error, but in this case we just skip infill generation entirely.

An example of the previous Pikachu with a 50% is shown in Figure 13.

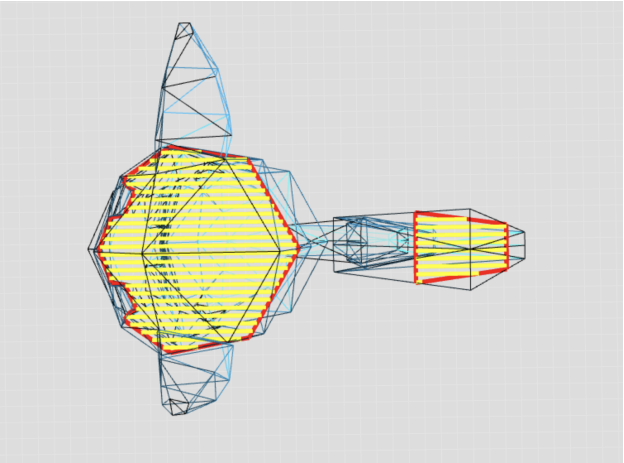


Figure 13: 50% infill density Pikachu

7.5 Polygon types

Having non-solid infill introduces some problems in edge cases. In case we are printing a cube, the first and last layer have to still be 100% infill. Furthermore, imagine we are printing the object represented in Figure 14.

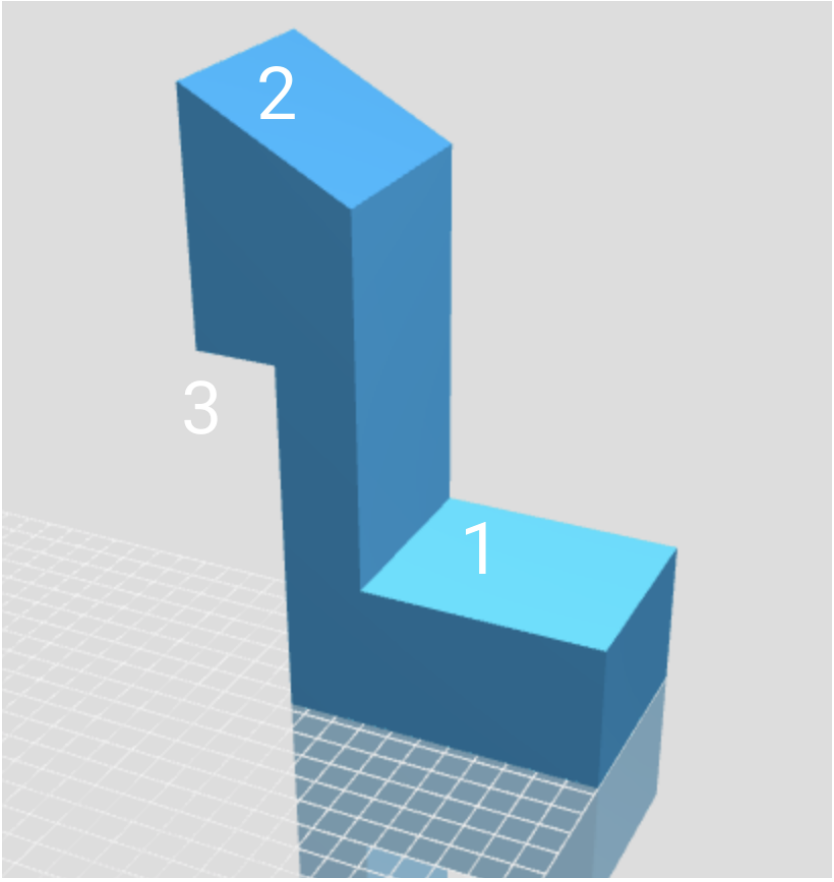


Figure 14: Top and Bottom parts in non top/bottom layers

We need to correctly identify (1) as a top surface, and (3) as a bottom surface, even though they are not in top or bottom layers.

Furthermore, (2) can be problematic: if the slope angle is close to horizontal, and the layer height is big enough, we will see infill, so we need to take that into account and generate top polygons there too.

All of these problems can be solved by polygon typing.

Let L be a layer. We will refer to the layer directly above L as L^1, and to the layer directly below L as L_1.

Every layer is simply a surface, but we cannot treat the whole layer the same. If we look at Figure 14 again, we can see that the layer containing the surface (1) is a rectangle. Technically, we will never slice this layer (as we will always make sure that our slicing plane does not contain any vertex of the model). However, lets call the layer immediately below L.

If we print L with infill settings, L^1 will not cover the top surface, and the infill pattern will be visible. We would like to solve this by printing the affected area with solid infill.

To detect this, we will take the difference $T = L \cap (L - L^1)$, our top polygons. Similarly, $B = L \cap (L - L_1)$ will be our bottom polygons. In particular, in case of the bottom layer, $B = L$, and in case of the top layer, $T = L$.

This feature is quite complex and expensive, so it will be left out in the current version of the slicer, but may be added in future work.

7.6 Ordering segments

Ordering segments is a simple task to do. We will simply order them by vertex proximity. This way we minimize travel time (when we are moving the print head but not extruding any plastic).

The ordering should be done for each polygon separately. Otherwise we will be traveling back and forth between polygons, greatly impacting efficiency, and also affecting the final print quality.

7.7 Toolpath generator

The only thing that's left before printing is to translate the segments into machine code. This is done by taking the start and end vertices of each segment and append the corresponding instructions to a list.

It might seem simple. Take the star and end of each segment and tell the printer to lay down some plastic in the path. There is even an instruction that does exactly that. In practice, it is not so simple and there are a lot of things to tune to get this right.

We will mainly use the `G1` command [25]. This command performs a straight line move between two points. We can control X, Y, Z axis, E for the extruder, and the speed.

Every time we finish all lines of a layer we will move the Z axis up by the layer height.

7.7.1 Calculating the amount of filament

We will assume that our printer drops a rectangular prism along each of the segments it extrudes. The width will be the same as the line width (usually 0.4mm), the height will be same as the layer height (0.06-0.3mm), and the length will be the segment's length itself. Let's call these w, h and l

Our material comes in a spool of filament, 1.75mm in diameter. For each segment we print, we need to calculate the length of this filament we want to push down through the extruder. We can calculate the volume that we need or a segment, V, as follows:

$$V = w * h * l$$

The area A of the filament will simply be:

$$A = \pi * (1.75/2)^2$$

Then, the extrusion length e will simply be:

$$e = V/A$$

7.7.2 The decimation problem

Let's say we have a high quality model with a large number of triangles. When we generate the segments, we will be slicing each of them and generating segments for each triangle. There will be a lot of segments that are extremely short.

When we feed the instructions to the printer, it will try to move a distance that is smaller than its accuracy, effectively extruding zero material while moving zero distance. This would not be a problem in an ideal world, where we can process instructions without consuming time.

In practice though, the printer will be executing instructions with little to no travel, and the overhead of calculating the position make it go too slow, creating a lot of problems. For example, since the material is melted, it will start to fall out of the extruder, jam it, start to burn, or just get stuck.

We can solve this by decimating the input model, but this does not guarantee that the output segments will be large enough for the printer, so a better approach is to decimate the output segments. Decimation of output segments is out of the scope of this project but worth mentioning because of its importance with high quality models.

7.7.3 The oozing problem

When we extrude filament everything is fine. We move from A to B while laying down material. However, we will find some cases where we have to travel without extruding. In this case, since we are working with a molten material, we will have oozing of the material.

We can easily prevent oozing with a technique called retraction.

Right before traveling, we will drive the extruder motor backwards a few mm, which will create suction in the hot end. This will prevent the molten material from oozing out of it.

Then, when we arrive at a point where we want to print again, we just reverse this distance again. This process requires some fine tuning depending on the printer, the distance traveled, the nozzle diameter, filament size, and travel speed. Since this is tricky to get exactly right, the idea was to leave it out, but in the end it has been implemented anyway.

8 Benchmarks

We will be comparing OpenSlicer against existing slicer software that have been around for a while.

- Cura (2014), is a commercial slicer created by Ultimaker, one of the largest 3D printing companies in the world.
- Slic3r (2011), created by Alessandro Ranellucci as an open source project, is maintained by a large community of developers and 3D printing enthusiasts.

Our benchmarks will provide a good understanding of the large difference between a product that has been developed in a few months by a single person and a commercial product that has been around for years.

To avoid any potential benchmark inaccuracy caused by differences in print settings, all tests will be printed with the configuration shown in Table 11

Setting	Value
3D Printer	Original Prusa i3 MK3
Material	PET-G
Nozzle temperature	240°C
Bed temperature	76°C
Layer height	0.2mm
Speed	40mm/s
Infill percentage	100

Table 11: Benchmark print settings

We will also use no brims, rafts, skirts, or supports.

We will use the previously shown Pikachu model, which will serve just fine for slicer benchmarks.

8.1 Slicing Performance

The results of the Pikachu model are shown in Table 12.

Slicer	Slicing time (s)
Cura	6.05
Slic3r	3.63
OpenSlicer	12.89

Table 12: Slicing time benchmark results for the Pikachu model

Given the relative slowness of the browser, these results are surprisingly good.

8.2 Output file size

Just for the sake of completeness, in Table 13 we show the generated file sizes by the different slicers.

Slicer	Output file size (MB)
Cura	1.8
Slic3r	1.3
OpenSlicer	0.81

Table 13: G-code output file size

Cura and Slic3r generate a similar output size, and take into account a vast number of details which result in an increased file size.

Since OpenSlicer does not have things like slowing down before a sharp corner, changing fan speeds, z-hop, brims, skirts, speed changes depending on the segment length, etc, we get a lot smaller files.

A smaller file size in this case is a bad thing, and indicates a lack of quality, settings and considerations.

8.3 Printing time

We will be printing the Pikachu model sliced previously to compare printing times. Results are shown in Table 14.

Slicer	Print time (minutes)
Cura	41m 32s
Slic3r	58m 29s
OpenSlicer	1h 5m 30s

Table 14: Print time benchmark

For this test, we can see that Cura and Slic3r differ significantly in their print times. However the end result is very similar. Both slicers produce a good quality, strong and accurate print.

OpenSlicer takes a bit longer than Slic3r, but the end quality is considerably worse than that of both Cura and Slic3r. This is simply a lack of time. It is not realistic to produce a high quality print in the time that this project is built.

8.4 Dimensional accuracy

For this test we will print a 10mm cube. Results are shown in Table 15.

Slicer	Max error (mm)
Cura	0.05
Slic3r	-0.03
OpenSlicer	0.22

Table 15: Dimensional accuracy benchmark

The measurements above are the average of 10 cubes per slicer, with different rotations and positions.

The reason why OpenSlicer has such a large error is very simple to explain, because currently we are printing exactly on the perimeter of the object. Plastic will flow both ways, creating an error of half the line width. This error accounts for 0.2mm and the real dimensional accuracy of OpenSlicer should be 0.02mm once this is fixed.

The errors are pretty close to zero, and the small variations are very likely simple measurement precision limitations, so we can conclude that all three slicers have the same dimensional accuracy.

8.5 Quality and surface finish

For this benchmark we will print the Pikachu model again, once with Slic3r, and once with OpenSlicer.

The results are shown in Figure 15.

Figure 15: Slic3r and OpenSlicer versions of the Pikachu model

Cura's version is omitted because it is looks exactly the same as Slic3r's version.

We can conclude from this experiment that OpenSlicer has a long way to go in terms of surface quality and oozing.

9 Future work

There is plenty of room to make improvements, considering this project is extremely basic and just performs the simplest slicing tasks.

Top and bottom detection are essential for a quality slicer. This will avoid gaps in slopes that are close to horizontal. This feature can be implemented easily by taking the difference of a layer with it's next layer.

Then, support structure generation, to avoid objects printing in mid-air. This improvement is easy to implement once top and bottom detection is implemented, as it consists of basically the same steps, but using a different pattern in the generation part.

After that, multiple objects support is a nice feature to have, and really simple too. We can just intersect each layer with multiple objects at the same time, and from there on the slicer would be exactly the same.

It would be nice to add some settings that control the angle of the infill to reduce stress on the motors. Printing at a 45 degree angle is better than parallel to the X or Y axis, since it will balance the stress across both motors at the same time, wearing them out evenly. Printing at a 45 degree speed also improves print speed, because we can travel in both directions at the same time.

Multiple extruders, per part settings, variable layer height, multi-material support, and a lot of other features can be added, but are quite complex to implement.

10 Conclusions

Creating a slicer requires a deep understanding in multiple areas. It is extremely hard to get good results with a single person in a limited time span.

Using JavaScript significantly impacts performance and makes extremely large objects slow to slice.

Slicing is a highly parallelizable process which can take advantage of multiple cores of the CPU. JavaScript restricts us to a single core, which is a great impact on the overall performance of the program.

Browsers impose limits on the amount of memory a page can use, which, in turn, limits our slicer to small to mid-sized objects.

Having a slicer on the phone allows us to inspect a model before slicing it on a more powerful desktop slicer.

It is convenient to have a slicer on the phone, and the ability of modern printers to upload GCODE via a web interface allows us to control the whole process from our phone. This is especially interesting in on-the-go scenarios, where we have no access to a computer.

Having a slicer in JavaScript allows us to have a platform independent code base, as well as make deployment and update management extremely simple.

This project has been a fun way to learn more about both simple and advanced concepts in geometry, math, graphics, and 3D printing in general.

Initially I thought it would be a simple task to develop a basic slicer, but after actually doing it, I realize that it is more complex than it seemed, and now I understand why after almost a decade, existing slicers are still under active development.

Although there is room for improvement, I am satisfied with the end result, and the fact that I actually got objects to print successfully.

11 References

[1] Ricardo Cabello. three.js, 2019. threejs.org.

[2] Lingjun Meng. Boolean operations and offsetting library in Javascript, 2016. github.com.

[3] Cristian Pallarés. An open source freeware polygon clipping library, 2017. github.com.

[4] Mapbox. A very fast JavaScript polyline and polygon clipping library, 2018. github.com.

[5] Ultimaker. CuraEngine, 2019. github.com.

[6] Alessandro Ranellucci. Slic3r, 2019. slic3r.org.

[7] Freedman, Michael H.; Quinn, Frank (1990). Topology of 4-Manifolds, 1990. Princeton University Press. ISBN 0-691-08577-3.

[8] The git community. git-bisect, 2019. git-scm.com.

[9] JetBrains s.r.o. WebStorm, 2019. jetbrains.com.

[10] Smartapp.com, Inc. Gantter, 2019. gantter.com.

[11] Glassdoor, Inc. Software Developer Salaries in 2018. glassdoor.com.

[12] Glassdoor, Inc. Project Manager Salaries in 2018. glassdoor.com.

[13] Glassdoor, Inc. QA Engineer Salaries in 2018. glassdoor.com.

[14] Endesa S.A. Latest Electricity and Gas Rates. endesaclientes.com.

[15] Vodafone España S.A.U. Precios Internet y Fijo. vodafone.es.

[16] Nespresso S.A. Cápsulas de café. nespresso.com.

[17] Science Focus. Human breathing contribution to climate change. sciencefocus.com.

[18] Hitachi, Ltd. Production process for polylactic acid (PLA). hitachi.com.

[19] 3D Insider. The Environmental Impact of 3D Printing, 2017. 3dinsider.com.

[20] Khronos Group. OpenGL ES for the Web. khronos.org.

[21] Xueqiao, J., Cabello, R., et al. OrbitControls. github.com.

[22] Leeper, A., Cabello, R., et al. STL Loader. github.com.

[23] The Bootstrap Team. Bootstrap 4. getbootstrap.com.

[24] Agustín "Flowalistik". Low-poly Pikachu. thingiverse.com.

[25] Marlin. Gcode documentation. marlinfw.org.